Device Tree詳解

Yoichi Yuasa

OSAKA NDS Embedded Linux Cross Forum #8

自己紹介

- 湯浅陽一
- 1999年よりLinux kernel開発に参加
- MIPSアーキテクチャのいくつかのCPUへLinux kernelを移植

本日の内容

- Device Treeの概要
- デバイスドライバの機能拡張にDevice Treeで対応する方法の説明

Device Treeとは

- Open Firmware Device Tree
- Open Firmwareでハードウエア情報をやり取りするためのデータ構造

Device Treeとは(続き)

- 中身はハードウエア構成情報
  - CPU情報
  - バス接続構成情報
  - アドレス&サイズ情報
  - 割り込み情報
  - クロック情報
  - デバイス有効/無効情報
  - その他デバイス特有情報など
- LinuxではPowerPCで最初にサポート

Device Treeのスタイル

```
/ {
    node1 {
        a-bool-property;
        a-string-property = "A string";
        a-string-list-property = "first string", "second string";
        a-byte-data-property = [0x01 0x23 0x34 0x56];
        child-node1 {
            first-child-property;
            second-child-property = <1>;
            a-string-property = "Hello, world";
        };
        child-node2 {
        };
    };
    node2 {
        an-empty-property;
        a-cell-property = <1 2 3 4>; /* each number (cell) is a uint32 */
        child-node1 {
        };
    };
};
```

Open Firmware

- SunのSPARCベースマシン、IBMのPOWERベースマシン、AppleのPowerPCベースマシンなどで利用
- ハードウエアに依存しないファームウエア
  - Open Firmware上で動作するプログラムは
    バイトコード(FCode)を利用
- IEEE 1275-1994として標準化

Linux ArmでDevice Treeを導入した背景

- 2000年前半はArmではATAGを利用
- ATAGはリスト形式のデータ
- データ例

Tag name	Value	Size	Description
ATAG_CORE	0x54410001	5(2 if empty)	Start list(flags, pagesize, rootdev)
ATAG_MEM	0x54410002	4	Physical area of memory
ATAG_SERIAL	0x54410006	4	Board serial No.
ATAG_NONE	0x00000000	2	End list

…

Linux ArmでDevice Treeを導入した背景(続き)

- ATAGにプラスの対応
  - ボード毎のデータをソースコードにハードコーディング
  - ボード毎にカーネルコンフィグレーションを用意
- サポートするボード数が増えると破綻が見えてきた
  - 1つのボードをサポートするために複数ファイルを追加
  - Arm開発者がボードサポートに本腰を入れてボード数が急激に増加
- 既存でサポートされているOpen Firmware Device Treeを採用

Device Treeの利用

- Device Treeソースからコンパイルしてバイナリ(dtb)を生成
  dtc -O dtb -o foo.dtb foo.dts
- dtb単体またはLinux kernelとdtbを結合して利用可能
  - ブートローダでdtbをメモリ上に配置
  - 起動したkernelからDevice Treeを参照
  - 結合した状態でもLinux kernelと同時にロードされるだけで動作は同一
- dtbを入れ替えれば容易に複数のボードを1つのLinux kernel バイナリ(カーネルコンフィグレーション)でサポートできる

Device Treeの問題点

- Linux kernelのサイズが大きくなる
  - 1つのkernelで複数ボードをサポートする場合
  - kernelが大きくなりやすいので小さなシステムで利用するにはカーネルコンフィグレーションのチューニングが必要

デバイスドライバの機能拡張とDevice Tree

- 個別機能のON/OFF
  - ソースコードを変更してON/OFFするとkernelのコンパイルが必要
- デバイス設定値をボード毎に細く調整
  - ソースコードを変更して対応するとkernelのコンパイルが必要
  - 管理するkernelバイナリの数が増える
- Device Treeだけを変更して機能を切り替える

Device Tree propertyをデバイスドライバで読み出す

- デバイスドライバのprobe関数内でpropertyを読み出す

```
static struct platform_driver foo_driver = {
    .driver = {
        .name = “foodev”,
        .of_match_table = foo_id_table,
    },
    .probe = foo_probe,
    .remove = foo_remove,
};
```

個別機能をON/OFFする (Device Tree)

- Bool propertyを利用
- Device Treeのデバイスnode内にBool propertyが有るか無いかで動作を切り替える

```
foodev: foodev0 {
    compatible = “foodev”;
    foodev-barfunc-enable;
};
```

個別機能をON/OFFする (デバイスドライバ)

- Device Treeのpropertyを読み出して設定を切り替える

```
sturct device *dev;
if (of_property_read_bool(dev->of_node, “foodev-barfunc-enable”)) {
    個別機能enable処理
} else {
    個別機能disable処理or何もしない
}
```

デバイス設定値をボード毎に細かく調整(Device Tree)

- 数値propertyを利用
- Device Treeのデバイスnode内に数値propertyが有ればその値を読み出し設定する

```
foodev: foodev0 {
    compatible = “foodev”;
    foodev-barvalue = <123>;
};
```

デバイス設定値をボード毎に細かく調整(デバイスドライバ)

- Device Treeのpropertyを読み出して値を利用

```
sturct device *dev;
u32 value;
int ret;
ret = of_property_read_u32(dev->of_node, “foodev-barvalue”, &value)) ;
if (ret) {
    エラー処理
}
```

valueの値をデバイスへ設定

まとめ

- Device TreeからBoolや数値以外にも文字列や数値/文字列の配列なども取得できる
- Device Treeから値を取得して利用することによりソースコードの値をいちいち書き換えるより柔軟性が増す

参考

- Documentation/arm/Booting
- http://www.simtec.co.uk/products/SWLINUX/files/booting_article.html
- https://elinux.org/Device_Tree_Reference